Incremental Sparsification for Real-time Online Model Learning

Duy Nguyen-Tuong
Max Planck Institute for Biological Cybernetics
72076 Tübingen, Germany
duy.nguyen-tuong@tuebingen.mpg.de

Jan Peters
Max Planck Institute for Biological Cybernetics
72076 Tübingen, Germany
jan.peters@tuebingen.mpg.de

Abstract

Online model learning in real-time is required by many applications such as in robot tracking control. It poses a difficult problem, as fast and incremental online regression with large data sets is the essential component which cannot be achieved by straightforward usage of off-the-shelf machine learning methods (such as Gaussian process regression or support vector regression). In this paper, we propose a framework for online, incremental sparsification with a fixed budget designed for large scale real-time model learning. The proposed approach combines a sparsification method based on an independence measure with a large scale database. In combination with an incremental learning approach such as sequential support vector regression, we obtain a regression method which is applicable in real-time online learning. It exhibits competitive learning accuracy when compared with standard regression techniques. Implementation on a real robot emphasizes the applicability of the proposed approach in real-time online model learning for real world systems.

1 Introduction

In recent years, model learning has become an important tool in a variety of robotics applications such as terrain modeling, sensor modeling (Plagemann *et al.*, 2007), model-based control (Nakanishi and Schaal, 2004; Schaal *et al.*, 2002) and many others. The reason for this rising interest is that due to the increasing complexity of modern robot systems, accurate analytical models are often hard to obtain. Model learning can be a useful alternative as the model is estimated directly from measured data. Unknown nonlinearites are directly taken in account, while they are neglected either by the standard physics-based modeling techniques or approximated by hand-crafted models. Nevertheless, the excessive computational complexity of the more advanced regression techniques still hinders their widespread application in robotics. Models that have been learned offline can only approximate the model correctly in the area of the state space that is covered by the sample data, and often do not generalize beyond that region. Thus, in order to cope with unknown state space regions online model learning is essential. Furthermore, it also allows the adaptation of the model to changes in the robot dynamics, unforeseen load or time-variant torque generation of the actuators. However, real-time online model learning poses three major challenges: first, the learning and prediction process needs to be sufficiently fast; second, the learning system needs to deal with very large amounts of data; third, the data arrives as a continuous stream and thus the model has to be continuously adapted to new training examples over time. Several approaches for real-time model learning for robotics have been introduced, such as locally weighted projection regression (Vijayakumar *et al.*, 2005) or local Gaussian process regression (Nguyen-Tuong *et al.*, 2008). In these methods, the state space is partitioned in local regions for which local models are approximated and thus these methods will not make use of the global behavior of the embedded functions. As the proper allocation of relevant areas of the state space is essential, appropriate online clustering becomes a central problem of these approaches. For high dimensional data, partitioning of the state space is well-known to be a complicated problem (Vijayakumar *et al.*, 2005; Nguyen-Tuong *et al.*, 2008). To circumvent this online clustering, as done in (Nguyen-Tuong *et al.*, 2008; Vijayakumar *et al.*, 2005) for example, an alternative is to find a sparse representation of the *known* data

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

(Rasmussen and Williams, 2006; Schölkopf and Smola, 2002). For robot applications, however, it requires finding an incremental sparsification method applicable in real-time online learning – a major challenge tackled in this paper.

Inspired by the work in (Engel *et al.*, 2004; Schölkopf *et al.*, 1999), we propose a framework for incremental, online sparsification which can be integrated into an existing online learning method, making it applicable for model learning in real-time. The suggested sparsification is performed using a test of linear independence to select a sparse subset of the training data points, often called the *dictionary*. The resulting framework allows us to derive criteria for incremental insertion and deletion of dictionary data points, which are two essential operations in this online learning scenario. Since the dictionary size needs to be small for real-time learning, we extend the dictionary by a large scale database enabling online learning for larger amounts of data. For evaluation, we combine our sparsification framework with sequential support vector regression (Vijayakumar and Wu, 1999) for online learning of the inverse dynamics model (Spong *et al.*, 2006). The rest of the paper will be organized as follows: first, we present our sparsification framework which enables real-time online model learning. Subsequently, the efficiency of the proposed approach in combination with sequential online support vector regression (Vijayakumar and Wu, 1999) (SOSVR) is demonstrated by a comparison of learning inverse dynamics models with well-established regression methods, i.e., ν-support vector regression (ν-SVR) (Schölkopf *et al.*, 2000), Gaussian process regression (GPR) (Rasmussen and Williams, 2006), locally weighted projection regression (LWPR) (Vijayakumar *et al.*, 2005) and local Gaussian process regression (LGP) (Nguyen-Tuong *et al.*, 2008). Finally, the capability of SOSVR using online sparsification for real-time model learning will be illustrated by online approximation of inverse dynamics models for real-time tracking control on a Barrett WAM.

2 Incremental Sparsification for Large Scale Online Learning

Online model learning requires incremental updates, as the data arrives sequentially over time. There have been many attempts to develop incremental, online algorithms for kernel methods, such as incremental SVM (Cauwenberghs and Poggio, 2000), sequential SVR (Vijayakumar and Wu, 1999) or the kernel recursive least-squares algorithm (Engel *et al.*, 2004) (for an overview see (Schölkopf and Smola, 2002)). However, most incremental kernel methods do not scale to online learning in real-time (e.g., for online learning with model updates at 50 Hz or faster), since they are neither sparse (Cauwenberghs and Poggio, 2000; Vijayakumar and Wu, 1999) (as they use the complete data set for model training), nor do they restrict the size of the sparse set (Engel *et al.*, 2004). For large scale learning, small sparse sets often do not suffice for a competitive learning performance. To overcome these shortcomings, we propose the setup shown in Figure 2.

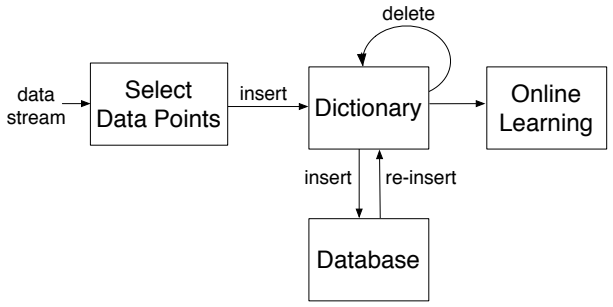


Figure 2: Online Sparsification with Database.

To efficiently make use of the stream of continuously arriving data, we select the most informative data points for the dictionary. To cope with the computational cost and to ensure real-time constraints, a fixed budget for the dictionary can be determined by both intended learning speed and available computational power. If this hard limit is reached, dictionary points will need to be deleted. In order to make the sparse method appropriate for large scale learning, the dictionary is additionally supported by a database retaining a larger amount of dictionary points. Finally, for the model training using dictionary data, online kernel regression methods can be employed, e.g., sequential SVR (Vijayakumar and Wu, 1999) or incremental SVM (Cauwenberghs and Poggio, 2000). Inspired by the work in (Engel *et al.*, 2004; Schölkopf *et al.*, 1999), we use a linear independence measure to select the most informative points given the current dictionary. Based on this measure, we derive criteria to remove data points from the dictionary, if a given limit is reached. For the insertion of data points from database to the dictionary, we select a limited number of database points which are *closest* to the most recent query point. This operation can be performed efficiently for large data sets using nearest neighbor search (Muja, 2009).

2.1 Test of Linear Independence

At any point in time, our algorithm maintains a dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^m$. To test whether a new point $\{\mathbf{x}_{m+1}, y_{m+1}\}$ should be inserted into the dictionary, we need to ensure that it can not be approximated in

the feature space of the current dictionary set. This test can be performed using a value δ defined as

$$\delta = \left\| \sum_{i=1}^{m} a_i \mathbf{\Phi}(\mathbf{x}_i) - \mathbf{\Phi}(\mathbf{x}_{m+1}) \right\|^2 , \quad (1)$$

(see, e.g., (Engel *et al.*, 2004; Schölkopf and Smola, 2002)), where $\mathbf{\Phi}$ is the feature vector and a_i denote the coefficients of linear dependence. The coefficients a_i can be determined by minimizing δ, which yields the optimal coefficient vector $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$. Here, $\boldsymbol{K} = \mathbf{k}(\mathbf{X}_m, \mathbf{X}_m)$ is the kernel matrix evaluated for the dictionary data set $\mathbf{X}_m$, and $\boldsymbol{k} = \mathbf{k}(\mathbf{X}_m, \mathbf{x}_{m+1})$ is the kernel vector. After substituting the optimal value $\boldsymbol{a}$ into Equation (1), δ becomes

$$\delta = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1}) - \boldsymbol{k}^T \boldsymbol{a} . \quad (2)$$

The value δ is an independence measure indicating how well a new data point $\mathbf{x}_{m+1}$ can be approximated in the feature space of a given data set. Thus, the larger the value of δ is, the more *independent* is $\mathbf{x}_{m+1}$ from the dictionary set $\mathbf{X}_m$, and the more *informative* is $\mathbf{x}_{m+1}$ for the learning procedure. Using δ we can decide whether to insert new data points into the dictionary. The sparsification procedure is summarized in Algorithm 2. The independence measure δ as given in Equation (2), can be used as a criterion for selecting new data points by thresholding (see Algorithm 2). The threshold parameter η implicitly controls the level of sparsity. However, for a given threshold value η, the number of dictionary points selected from the online data stream is *not* known beforehand and thus can be very large. Large dictionaries are prohibitively expensive in terms of computational complexity and not real-time capable. To cope with this problem, we define an upper bound on the dictionary size and delete old dictionary points if this bound is reached.

For deleting old dictionary points, we consider the independence measures δ^i of every dictionary point i. The value δ^i indicates, how well the dictionary point i can be approximated by the remainder of the dictionary. The score δ^i has to be updated, when a new data point is inserted into the dictionary or an old data point is removed from the dictionary. In Sections 2.2 and 2.3, we will discuss an incremental way of updating the value δ^i applicable for real-time online computation.

Algorithm 1: Independence test with online updates of the dictionary.

Input: new point $\{\mathbf{x}_{m+1}, y_{m+1}\}$, η, MaxNr.
Compute: $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$ with $\boldsymbol{k} = \mathbf{k}(\mathbf{X}_m, \mathbf{x}_{m+1})$
Compute: $\delta = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1}) - \boldsymbol{k}^T\boldsymbol{a}$
if $\delta > \eta$ **then**
  if number of dictionary points < MaxNr **then**
    Update the dictionary using Algorithm 2 to insert $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
  else
    Update the dictionary using Algorithm 3 to insert $\{\mathbf{x}_{m+1}, y_{m+1}\}$ and to replace an old dictionary point.
  end if
  Online model training using the new dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^{m+1}$.
end if

2.2 Insert a New Point into the Dictionary

After inserting a new data point $\mathbf{x}_{m+1}$ into the dictionary, we have to incrementally update the independence measure δ^i for *every* dictionary point i, as changing the dictionary implies a change for δ^i. This update for an existing dictionary point $\mathbf{x}_i$ is done by adjusting the corresponding coefficient vector $\boldsymbol{a}^i$. Updating $\boldsymbol{a}^i$ implies an update of $(\boldsymbol{K}^i)^{-1}$ and $\boldsymbol{k}^i$. Here, inserting a new point will extend $\boldsymbol{K}^i$ by a row/column and $\boldsymbol{k}^i$ by a value, respectively, such that

$$\boldsymbol{K}^i_{\mathrm{new}} = \begin{bmatrix} \boldsymbol{K}^i_{\mathrm{old}} & \boldsymbol{k}_{m+1} \\ \boldsymbol{k}^T_{m+1} & k_{m+1} \end{bmatrix}, \quad \boldsymbol{k}^i_{\mathrm{new}} = \begin{bmatrix} \boldsymbol{k}^i_{\mathrm{old}} \\ k_{i,m+1} \end{bmatrix}, \quad (3)$$

where $k_{m+1} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$, $k_{i,m+1} = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_{m+1})$, $\boldsymbol{k}_{m+1} = \mathbf{k}(\mathbf{X}^i_m, \mathbf{x}_{m+1})$ with $\mathbf{X}^i_m = \mathbf{X}_m \backslash \{i\}$. Using the Equation (3), the incremental update of the inverse matrix $(\boldsymbol{K}^i_{\mathrm{new}})^{-1}$ is given by

$$\left(\boldsymbol{K}^i_{\mathrm{new}}\right)^{-1} = \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i (\boldsymbol{K}^i_{\mathrm{old}})^{-1} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T & -\boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T & 1 \end{bmatrix} . \quad (4)$$

This derivation leads to the update rule for the linear independency value δ^i for the i-th dictionary point given by

$$\delta^i = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) - \boldsymbol{k}^{iT}_{\mathrm{new}} \boldsymbol{a}^i_{\mathrm{new}} , \quad \text{with}$$

$$\boldsymbol{a}^i_{new} = \frac{1}{\gamma_i} \begin{bmatrix} \gamma_i \boldsymbol{a}^i_{\mathrm{old}} + \boldsymbol{\alpha}_i \boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\mathrm{old}} - k_{i,m+1} \boldsymbol{\alpha}_i \\ -\boldsymbol{\alpha}_i^T \boldsymbol{k}^i_{\mathrm{old}} + k_{i,m+1} \end{bmatrix} . \quad (5)$$

The variables γ_i and $\boldsymbol{\alpha}_i$ are determined by $\boldsymbol{\alpha}_i = (\boldsymbol{K}^i_{\mathrm{old}})^{-1} \boldsymbol{k}_{\mathrm{m+1}}$ and $\gamma_i = k_{m+1} - \boldsymbol{k}^T_{m+1} \boldsymbol{\alpha}_i$. The procedure for insertion of new data points with δ-update is summarized in Algorithm 2.

2.3 Replacing Dictionary Points

As the data arrives continuously in an online setting, it is necessary to limit the number of dictionary points so that the computational power of the system is not exceeded and the real-time constraints are not violated. The deletion can be performed straightforwardly by deleting a dictionary point whose independency value is *minimal*. The idea is to delete points which are most

Algorithm 2: Update the dictionary after inserting a new data point.

Input: new dictionary point $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
Update dictionary $\{\mathbf{x}_i, y_i\}_{i=1}^{m+1}$
for $i=1$ **to** number of dictionary points **do**
  Compute
    $\boldsymbol{k}_{m+1} = \mathbf{k}(\mathbf{X}_m^i, \mathbf{x}_{m+1})$
    $k_{m+1} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$
    $k_{i,m+1} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_i)$
  Compute
    $\boldsymbol{\alpha}_i = \left(\boldsymbol{K}_{\text{old}}^i\right)^{-1} \boldsymbol{k}_{m+1}$
    $\gamma_i = k_{m+1} - \boldsymbol{\alpha}_i^T \boldsymbol{k}_{m+1}$
  Update δ^i as given in Equation (5).
  Update $(\boldsymbol{K}_{new}^i)^{-1}$ as given in Equation (4).
end for

dependent on other dictionary points, i.e. where the corresponding independence value δ^i is minimal.

Insertion and additional deletion of dictionary points also change the independence values of other dictionary points which subsequently have to be updated. Insertion of a new point with an additional deletion of the j-th dictionary point imply a manipulation of the j-th row/column of $\boldsymbol{K}^i$ and the j-th value of $\boldsymbol{k}^i$

$$\boldsymbol{K}_{new}^i = \begin{bmatrix} \boldsymbol{K}_{\text{old}(1:j)}^i & \boldsymbol{k}_{m+1(1:j)} & \boldsymbol{K}_{\text{old}(j:m)}^i \\ \boldsymbol{k}_{m+1(1:j)}^T & k_{m+1} & \boldsymbol{k}_{m+1(j:m)}^T \\ \boldsymbol{K}_{\text{old}(j:m)}^{iT} & \boldsymbol{k}_{m+1(j:m)} & \boldsymbol{K}_{\text{old}(j:m)}^i \end{bmatrix}, \quad (6)$$

$$\boldsymbol{k}_{\text{new}}^i = \begin{bmatrix} \boldsymbol{k}_{\text{old}(1:j)}^i & k_{i,m+1} & \boldsymbol{k}_{\text{old}(j:m)}^i \end{bmatrix}^T.$$

The values $\boldsymbol{k}_{m+1}$, k_{m+1} and $k_{i,m+1}$ are determined as shown in Section 2.2. The incremental update of the independence measure δ^i for every i-th dictionary point can be performed directly using an incremental matrix inverse update. Hence,

$$\delta^i = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_i) - \boldsymbol{k}_{\text{new}}^{iT} \boldsymbol{a}_{\text{new}}^i \quad \text{and} \quad \boldsymbol{a}_{\text{new}}^i = \boldsymbol{A}\, \boldsymbol{k}_{\text{new}}^i, \quad (7)$$

where $\boldsymbol{A}$ is computed by the update rule

$$\boldsymbol{A} = \boldsymbol{A}^* - \frac{\text{row}_j[\boldsymbol{A}^*]^T \boldsymbol{d}^T \boldsymbol{A}^*}{1 + \boldsymbol{d}^T \text{row}_j[\boldsymbol{A}^*]^T} \quad \text{with}$$

$$\boldsymbol{A}^* = (\boldsymbol{K}_{\text{old}}^i)^{-1} - \frac{(\boldsymbol{K}_{\text{old}}^i)^{-1} \boldsymbol{d}\, \text{row}_j[(\boldsymbol{K}_{\text{old}}^i)^{-1}]}{1 + \boldsymbol{d}^T \text{row}_j[(\boldsymbol{K}_{\text{old}}^i)^{-1}]^T}.$$

Here, $\boldsymbol{d}$ is determined by $\boldsymbol{d} = \boldsymbol{k}_{m+1} - \text{row}_j[\boldsymbol{K}_{\text{old}}^i]^T$ and row_j denotes the j-th row of a given matrix. The update of $(\boldsymbol{K}_{\text{new}}^i)^{-1}$ can be given as $(\boldsymbol{K}_{\text{new}}^i)^{-1} = \boldsymbol{A}$. The complete procedure is summarized in Algorithm 3.

2.4 Extending the Dictionary with a Database

Typically, only a small budget, e.g. 300 data points, can be maintained during online learning in real-time.

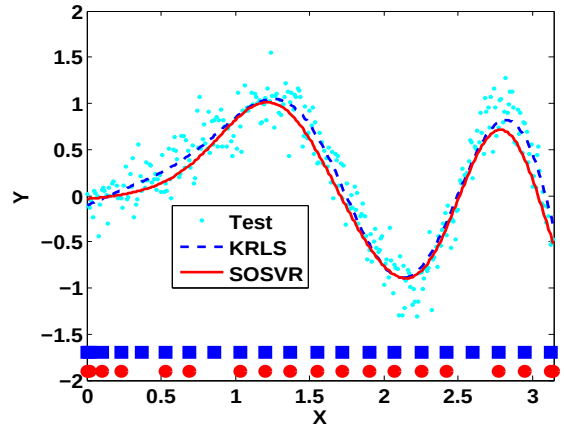
Algorithm 3: Replace the least important data point in the dictionary by a new one.

Input: new dictionary point $\{\mathbf{x}_{m+1}, y_{m+1}\}$.
Find dictionary point with minimal δ
  $j = \min_i \delta$
Update dictionary by overwriting point j:
  $\{\mathbf{x}_j, y_j\} = \{\mathbf{x}_{m+1}, y_{m+1}\}$
for $i=1$ **to** number of dictionary points **do**
  Compute
    $\boldsymbol{k}_{m+1} = \mathbf{k}(\mathbf{X}_m^i, \mathbf{x}_{m+1})$
    $k_{m+1} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_{m+1})$
    $k_{i,m+1} = \mathrm{k}(\mathbf{x}_{m+1}, \mathbf{x}_i)$
  Compute
    $\boldsymbol{d} = \boldsymbol{k}_{m+1} - \text{row}_j[\boldsymbol{K}_{\text{old}}^i]^T$
  Update δ^i as given in Equation (7).
  Update $(\boldsymbol{K}_{\text{new}}^i)^{-1} = \boldsymbol{A}$.
end for

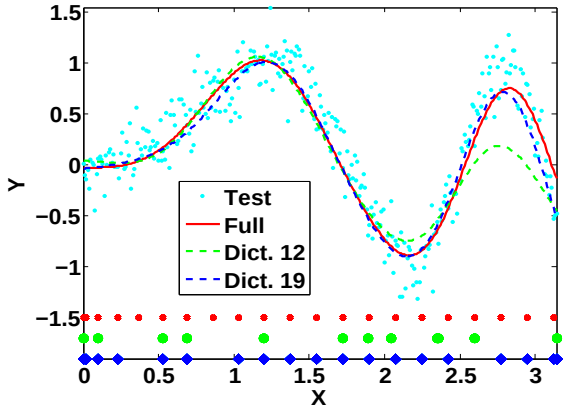
As a result, the sparse set has to be continuously updated by insertions and deletions of data points. While this way of constructing the dictionary ensures the real-time capability, it would entail an irretrievable loss of information. As changing the dictionary data over time implies a change of relevant features describing the recent state space region, *known* state space regions may have to be "relearned" from scratch. Furthermore, removing features from the sparse set depending on the current state space area can also lead to worse performances, as these features may contain relevant information for future state space regions. To overcome this limitation, we adopt a database retaining all previously removed dictionary points as illustrated in Figure 2. When a new data point is inserted in the dictionary, we check the neighborhood around this new point. If this region has been visited earlier, we pick the data points from its neighborhood out of the database. This approach provides the dictionary with data points *beforehand* if a known region is revisited and thereby eliminates the need for unnecessary and costly model relearning. Searching for k data points next to a given query point can be efficiently performed using k-nearest-neighbor search which yields a cost of $\mathcal{O}(k \log N)$, where N denotes the number of database points. Subsequently, we update the dictionary for these data points as shown in last Section 2.3. Note that the database is only active, when the deletion of dictionary points starts.

2.5 Comparison to Previous Work

Our work was inspired by the work in (Engel *et al.*, 2004) where they also used a linear independence measure to select dictionary points. However, they do not remove dictionary points again but instead show that the dictionary size is bounded for a given threshold η



(a) **Comparison with KRLS**



(b) **SOSVR prediction with different budgets**

Figure 3: Prediction performance after a single swept through the toy data set, where the data is incrementally fed to the algorithms. The dots show the positions of the corresponding dictionary points in the input space. **(a)** The performance of SOSVR using dictionary is quite similar to KRLS while the selection of the sparse data sets poses the main difference. **(b)** If the budget constraint is removed (i.e., using the full sparse set), SOSVR results in the same sparse solution as KRLS. Using a fixed budget, e.g., 12 or 19 dictionary points, the most relevant regions in the input space are covered with data points.

if the data space is assumed to be compact. In practice, for a given threshold value the actual dictionary size is data dependent and unknown beforehand, thus the dictionary can be very large. In order to cope with the computational constraints in real-time applications, the dictionary size has to be limited. Compared to the algorithm in (Engel *et al.*, 2004), our approach allows us to formulate an efficient insertion and deletion procedure for a given budget. Due to permanent dictionary update, the algorithm also allows fast model adaptation for unknown data. Considering the computational cost, we require $\mathcal{O}(m^3)$ for the dictionary update, where m is the number of dictionary points. In the following section, we will compare our approach with the kernel recursive least square (KRLS) (Engel *et al.*, 2004) using a toy data set. For the online model training, we combine our sparsification with the sequential online SVR (Vijayakumar and Wu, 1999) (SOSVR). It should be noted that other online model learning methods can also be used in combination with our incremental sparsification, for example incremental GP update as done in (Nguyen-Tuong *et al.*, 2008; Csato and Opper, 2002).

As a toy example, we generate a noisy data set where the relation between the target $\mathbf{y}$ and the input $\mathbf{x}$ is given by $y_i = \sin(x_i^2) + \varepsilon_i$. The data set consists of 315 points, where ε_i is white Gaussian noise with standard deviation 0.2 and x_i ranges from 0 to π. Here, the data is incrementally fed to the algorithms and the prediction is computed after a single swept through the data set. The results are shown in Figure 3, where $\eta = 0.001$ and a Gaussian kernel with width 0.4 is being used. In Figure 3 (a), it can be seen that the prediction performance of SOSVR using dictionary is quite similar to KRLS. Here, the selection of the sparse data points presents the main difference, while KRLS gradually fills up the input space with equally spaced data points. Figure 3 (b) shows the performance of SOSVR for different dictionary sizes. If we remove the budget constraint (i.e. *no* dictionary points are deleted), then SOSVR with sparsification results in the same sparse solution as KRLS, i.e., the dictionary points uniformly cover the input space. Using a fixed budget, the algorithm incrementally selects the sparse data points which represent the input space best. Our sparsification algorithm can be further improved for regression by additional consideration of the output space, as the predictive performance may also entail valuable information for the model learning.

3 Evaluations

In this section, we evaluate our sparsification approach used in combination with the sequential online SVR (SOSVR) in several different experimental settings with a strong focus on inverse dynamics modeling for robot tracking control.

First, we briefly review model-based control using inverse dynamics models and the necessity in learning these models from data. Subsequently, we evaluate our algorithm in the context of learning inverse dynamics. The learning accuracy of SOSVR using sparsification will be compared with other regression methods, i.e., LWPR (Vijayakumar *et al.*, 2005), GPR (Rasmussen and Williams, 2006), ν-SVR (Schölkopf *et al.*, 2000) and LGP (Nguyen-Tuong *et al.*, 2008). For this evaluation in inverse dynamics learning, we employ 2 data sets including synthetic data, as well as real robot data generated from the 7 degrees-of-freedom (DoF)

Barrett WAM, shown in Figure 4 (a). We further demonstrate that the online learning performance can be largely improved by supporting the dictionary with a large database. Finally, SOSVR with sparsification is applied for real-time online learning of inverse dynamics models for robot computed torque control.

3.1 Learning Inverse Dynamics for Control

The model-based tracking control law (Spong *et al.*, 2006) determines the joint torques $\mathbf{u}$ that are required for the robot to follow a desired trajectory $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, where $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$ denote the desired joint angles, velocity and acceleration. In computed torque control, the motor command $\mathbf{u}$ consists of two parts: a feedforward term $\mathbf{u}_{\text{FF}}$ to achieve the movement and a feedback term $\mathbf{u}_{\text{FB}}$ to ensure stability of the tracking. The feedback term can be a linear control law such as $\mathbf{u}_{\text{FB}} = \mathbf{K}_p\mathbf{e}+\mathbf{K}_v\dot{\mathbf{e}}$, where $\mathbf{e}$ denotes the tracking error with position gain $\mathbf{K}_p$ and velocity gain $\mathbf{K}_v$. The feedforward term $\mathbf{u}_{\text{FF}}$ is determined using an inverse dynamics model and, traditionally, the analytical rigid-body model is employed (Spong *et al.*, 2006). If a sufficiently precise inverse dynamics model can be approximated, the resulting control law $\mathbf{u}=\mathbf{u}_{\text{FF}}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)+\mathbf{u}_{\text{FB}}$ will drive the robot along the desired trajectory accurately.

One possibility to obtain an accurate inverse dynamics model is to learn it *directly* from measured data. The resulting problem is a regression problem that can be solved by learning the inverse dynamics model $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$ from sampled data and, subsequently, by using the resulting mapping for predicting the appropriate feedforward motor commands $\mathbf{u}_{\text{FF}}$. As trajectories and corresponding joint torques are sampled directly from the real robot, learning the inverse dynamics will include all nonlinearities of the system.

3.2 Offline Comparison in Learning Inverse Dynamics

For generation of the two data sets, i.e. one with simulation data and one with real robot data, we sample joint space trajectories and corresponding torques from an analytical model of the Barrett WAM, as well as from the real robot. This results in two test scenarios, each having 12000 training points and 3000 test points. Given samples $\mathbf{x}=[\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ as input and using the corresponding joint torques $\mathbf{y}=\mathbf{u}$ as targets, we have a regression problem with 21 input dimensions and 7 output dimensions (i.e., a single desired torque for each motor joint). The robot inverse dynamics model is estimated separately for each DoF employing LWPR, ν-SVR, GPR, LGP and SOSVR with sparsification.

Figure 4 (b) and (c) show the offline approximation errors on the test sets evaluated using the normalized mean square error (nMSE) which is defined as the fraction of mean squared error and the variance of target. For all data sets, the dictionary size of SOSVR is limited to be 3000 points. As can be observed from the results, SOSVR with sparsification is competitive in learning accuracy. In practice, it also shows that the models are easier to train using SOSVR with sparsification compared to local learning methods such as LWPR and LGP, as the latter require an appropriate clustering of the state space which is not straightforward to perform for many data sets.

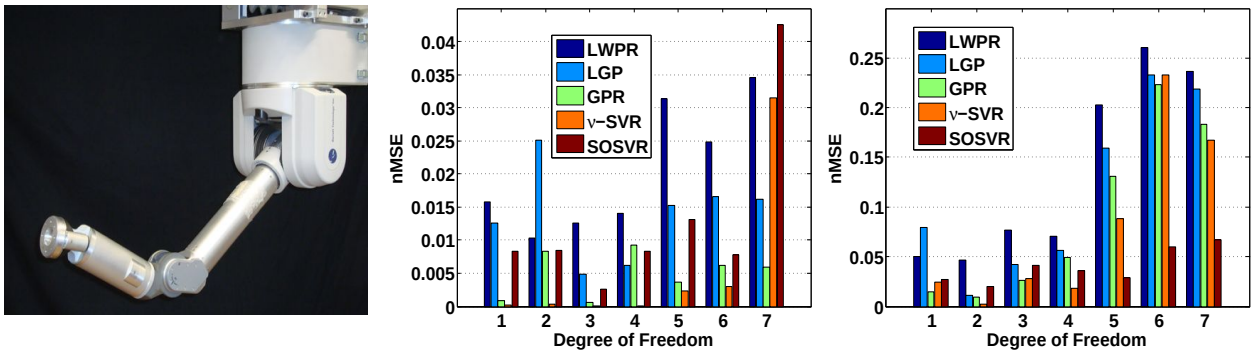
3.3 Model Online Learning with Database Extensions

For online learning in real-time, the dictionary size needs to be small in order to cope with the computational complexity and the real-time constraints. However, with a small dictionary the learning performance degrades for sufficiently complex movements. To overcome this problem, we extend our dictionary with a database. Figure 5 shows the online learning performance for the training data sets introduced in Section 3.2, i.e., simulated and real robot data. Here, the dictionary is limited to 50 data points, while the data is *incrementally* fed to the algorithm and the model is trained online.

As a new point is inserted to the dictionary, we pick 5 database points which are the nearest-neighbors to the most recent dictionary point and insert them as described in Sections 2.3 and 2.4. For the fast nearest neighbor search, we employ the library from Muja (2009). The results in Figure 5 show that the performance during online learning can be largely improved with a database support for a small dictionary size. Here, the prediction is performed online using the online trained model where the error is computed after a single sweep through the data set.

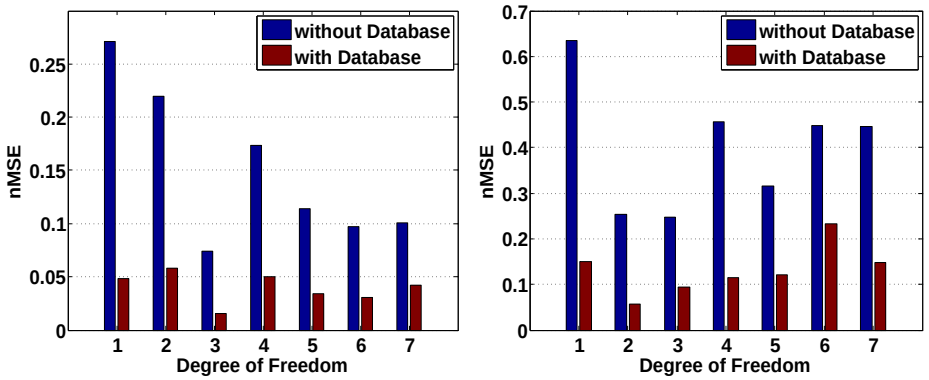
3.4 Real-time Model Online Learning in Computed Torque Control

In this section, we apply SOSVR with sparsification for the online learning of the inverse dynamics model for torque prediction in robot tracking control as introduced in Section 3.1. The control task with model online learning is performed in real time (500 Hz sample rate, i.e. every 2 ms we get a new test point) on the real Barrett WAM. In so doing, the trajectory $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ and the corresponding joint torques $\mathbf{u}$ are sampled online. The inverse dynamics model, i.e. the mapping $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \rightarrow \mathbf{u}$, is learned during the tracking task using the trajectory as input and the joint torque as target, starting with an empty dictionary. As the learned in-



(a) **Barrett WAM** (b) **Error on simulation data** (c) **Error on real WAM data**

Figure 4: **(a)** 7-DoF Barrett WAM, used for evaluations. **(b)** Error (nMSE) on simulated data from a robot model for every DoF. **(c)** Error (nMSE) on real robot data for every DoF. As shown by the results, SOSVR is competitive to standard batch regression methods such as ν-SVR and GPR, local learning methods such as LWPR and LGP. Especially for noisy data in **(c)**, sparsification can help to improve the learning performance, e.g., for the DoFs 5, 6 and 7. For SOSVR, the dictionary size is limited to be 3000 data points.



(a) **Error on simulation data in the online scenario** (b) **Error on real WAM data in the online scenario**

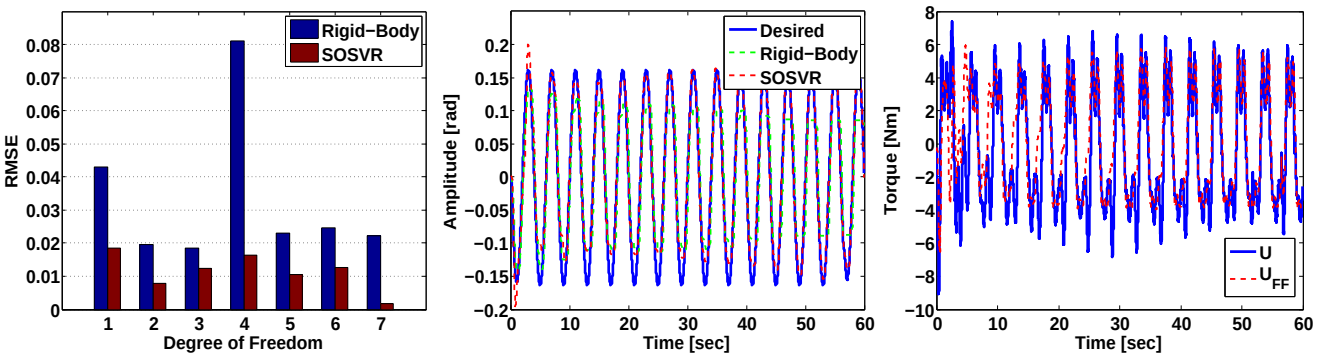
Figure 5: **(a)** Error (nMSE) on simulation data. **(b)** Error (nMSE) on real robot data for every DoF. As shown by the results, the online learning performance can be improved significantly by supporting the dictionary with a large scale database. Here, the dictionary size is limited to 50 data points, while the data is incrementally fed to the algorithm.

verse dynamics model is subsequently applied to compute the feedforward torques $\mathbf{u}_{\text{FF}}$, it can be observed that $\mathbf{u}_{\text{FF}}$ gradually converges against $\mathbf{u}$.

In this experiment, the maximal dictionary size is set to be 300, $\eta = 0.0001$ and a Gaussian kernel is used whose parameters are determined by cross-validation. The desired tracking trajectory is generated such that the robot's end-effector follows a 8-figure in the task space. For comparison, we also apply a rigid-body model for torque prediction (Spong *et al.*, 2006). The tracking results are shown in Figure 6, where tracking control task is performed for 60 *sec* on Barrett WAM. During this time, the dictionary is first incrementally filled up and, subsequently, updated about 700 times. Figure 6 (a) compares the tracking performances of the online learned model and the analytical model in joint space for all 7 DoFs, where the tracking error is computed as root mean square error (RMSE). It can be seen that the tracking accuracy can be largely improved, if the inverse dynamics model is approximated online. Figure 6 (b) shows the tracking performance in joint space for the 1. DoF, other DoFs are similar. It can be seen that the predicted torque $\mathbf{u}_{\text{FF}}$ (for 1. DoF) consistently converges to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning. The model approximation converges after around 40 *sec* resulting into improved tracking performance. A video showing the online model learning performance in real-time using sparsification can be found at "http://www.robot-learning.de". In that experiment, the dynamics model is modified online which is subsequently learned by the robot in real-time for tracking control using the proposed sparsification framework.

4 Conclusion and Future Work

Motivated by the need of fast online model learning for robotics, we develop an incremental sparsification



(a) **Tracking error on Barrett WAM in joint space for all 7 DoFs**

(b) **Tracking error on Barrett WAM in joint space for 1. DoF**

(c) **Feedforward u_{FF} and joint torque u of 1. DoF during SOSVR online learning**

Figure 6: **(a)** Tracking error in joint space after 60 *sec* computed as RMSE. Computed torque control with online learned model (SOSVR) outperforms common rigid-body model. **(b)** Tracking performance in joint space for the 1. DoF (other DoFs are similar). The result shows that the model converges after 40 sec with online learning. **(c)** Predicted feedforward torque and joint torque of the 1. DoF. The predicted torque $\mathbf{u}_{FF}$ converges consistently to the joint torque $\mathbf{u}$ as the latter is sampled as target for the model learning – as a result, the controls need to rely significantly less on feedback.

framework which can be used in combination with an online learning algorithm enabling an application in real-time online model learning. Our framework provides a way to efficiently insert and delete dictionary points taking in account the required fast computation during model online learning in real-time. Supported by a database, the approach also shows to be suitable for a large scale online learning. The implementation on a physical robot emphasizes the applicability in real-time online model learning for real world systems. Our future research will be focused on further extensions such as including the output space (e.g., the prediction errors) in the sparsification framework.

References

Cauwenberghs, G. and Poggio, T. (2000). Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems*.

Csato, L. and Opper, M. (2002). Sparse online gaussian processes. *Neural Computation*.

Engel, Y., Mannor, S., and Meir, R. (2004). The kernel recursive least-square algorithm. *IEEE Transaction on signal processing*, **52**.

Muja, M. (2009). Flann, fast library for approximate nearest neighbors. http://mloss.org/software/view/143/.

Nakanishi, J. and Schaal, S. (2004). Feedback error learning and nonlinear adaptive control. *Neural Networks*.

Nguyen-Tuong, D., Seeger, M., and Peters, J. (2008). Local gaussian process regression for real time online model learning and control. *Advances in Neural Information Processing Systems*.

Plagemann, C., Kersting, K., Pfaff, P., and Burgard, W. (2007). Heteroscedastic gaussian process regression for modeling range sensors in mobile robotics. *Snowbird learning workshop*.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. MIT-Press, Massachusetts Institute of Technology.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparameteric statistics for real-time robot learning. *Applied Intelligence*, pages 49–60.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT-Press, Cambridge, MA.

Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Müller, K.-R., Rätsch, G., and Smola, A. J. (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, **10**(5), 1000–1017.

Schölkopf, B., Smola, A., Williamson, R., and Bartlett, P. (2000). New support vector algorithms. *Neural Computation*.

Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2006). *Robot Dynamics and Control*. John Wiley and Sons, New York.

Vijayakumar, S. and Wu, S. (1999). Sequential support vector classifiers and regression. *International Conference on Soft Computing*.

Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*.